2

Towards building manageable multimedia network services

C. Aurrecoechea, A. A. Lazar and R. Stadler
Center for Telecommunications Research
Columbia University, New York, NY 10027
{cris, aurel, rolf}@ctr.columbia.edu

Abstract

The paper addresses the problem of designing and realizing manageable multimedia services. We introduce a generic object model which includes a set of cooperating objects that can be customized for a particular service and management requirements. These objects define the interaction between the service delivery and the service management systems and cover the aspects of instantiation of a service session, control of a session by the user, and management of sessions by the operator. By applying this model to the design of a multicast service, we enable the management system to monitor and control multicast sessions. To validate our approach, we have implemented the multicast service and the management functions on a high-performance emulation platform and we are currently porting the service to a broadband testbed. Finally, we argue that our model can be used to investigate designs for scalable and efficient systems that make possible to control the cost of management.

Keywords

Service Management, Multicast Service, Multimedia Services, Design Patterns, Software Prototyping, CORBA-based Management

1 INTRODUCTION

Until recently, telecommunication networks have been built upon a single technology and delivered as a small set of services. Because of the relative simplicity of the systems, the management task has focused primarily on the device level. Currently, we are witnessing an explosive growth in distributed multimedia applications and services, such as video conferencing, video-on-demand and collaborative distributed environ-

Management of Multimedia Networks and Services R. Boutaba & A. Hafid (Eds.)

ments, together with an increasing variety of underlying network technologies. As a result, telecom networks and environments are becoming more and more complex.

This situation has created major challenges for managing telecom multimedia environments. One of these challenges relates to the variety of services and technologies. Management architectures have been developed to address this problem. The TMN framework [ITU92], for instance, introduces an architecture that defines layers of increasing levels of abstractions (element management layer, network management layer, service management layer and business management layer). The service management layer deals with the services provided by the telecom system, including the multimedia services.

A second challenge is to design the new services (and technologies) in such a way that they can be managed on the appropriate level of abstraction. The fact is that most management systems today are built as an afterthought to service delivery systems. This makes it difficult to add the hooks for monitoring and controlling service activities later. More importantly, the service delivery system might have been designed and/or implemented differently, had the management requirements been taken into account during the analysis phase of the system's development.

This paper addresses the problem of designing and realizing manageable multimedia services. Our approach centers around defining a set of cooperating objects involved in the interaction between the service delivery and the service management system. The model we propose describes the interface between the two systems, and covers--in a generic way--the aspects of instantiation of a *service session*, its access by a user and its management by the operator. (A service session represents the information handled by all processes involved in providing a service; a session has a start, a duration and an end.) Our model suggests a generic solution to a recurring problem in service design. The model needs to be customized for a particular service and refined according to management requirements and available resources. In this sense, we are proposing a *design pattern* [GAM95] for making telecom services manageable.

This work focuses on service management in the sense of supervising the service delivery system. The management task includes monitoring and controlling the service delivery system to ensure that it operates as intended, maintaining the service-level agreements while achieving management objectives. The task is performed on a slower time-scale than the functions executing in the service delivery system. This activity relates to the OSI functional areas of performance and configuration management, and to the functions in the service management layer of the TMN architecture.

In the domain of service management we are considering, two activities can be identified: a) *managing the set of controllers* (i.e., routers, connection managers, etc.) which comprise the functionality of the service delivery system, and b) *managing the set of service instances* (such as audio multicast connections or VOD sessions) which are dynamically created, modified and terminated during the operation of the service delivery system. While we have presented some results on activity a) in [CHA96a], this paper focuses primarily on activity b), namely, on the management of service instances (or sessions) and aggregations of such instances.

We have applied our model for service management on a multicast VC service for a multiclass broadband network. To validate our approach and to gain experience, we have implemented the service and the management capabilities on two platforms, namely, a) on a high-performance emulation platform, which makes possible to develop a software prototype and to study its dynamic behavior and scaling properties in various scenarios [CHA96b], and b) a broadband testbed running xbind [LAZ96], a CORBA-based [OMG96] multimedia networking platform, on which services are fully implemented and the transport between multimedia devices is realized. In b) we use CORBA technology for building not only the service control system but also the service management system. CORBA provides a flexible object oriented environment for implementing distributed software systems, and it facilitates the integration of control and management software under a single technology.

The paper is organized as follows. Section 2 outlines the generic object model we propose for making services manageable. Section 3 introduces an object oriented design for a multicast service. In Section 4 the multicast service is made manageable based on our model. A first implementation of the system on the two platforms mentioned above is described in Section 5. Finally, Section 6 summarizes some of the experience we gained with this work and, more importantly, outlines research topics in service management that arise with the introduction of our model.

2 A GENERIC OBJECT MODEL FOR MANAGING SERVICE INSTANCES AND AGGREGATIONS

The service delivery system contains a set of interacting controllers that perform the tasks of service creation and service delivery in a distributed fashion [CHA96a]. Specifically, the system consists of a large number of controllers, with many instances of the same controller class spread throughout the system to support distributed operation and fault tolerance. A limited number of controller classes exist, each of which encapsulates a specific functionality, such as running a network control algorithm (admission control, routing, etc.). Most controllers are realized in software. They can be designed as communicating active objects. They are multi-threaded, i.e., they have separate address spaces, and they communicate by exchanging messages, which allows them to run on different time scales. Furthermore, they are designed as persistent objects, which are created during the initialization phase of the control system.

Our approach for making a service manageable introduces a new type of controller in the service control system: a dynamic object that is created for each new session. This object, which represents a service session (or service instance), makes itself accessible to the management system by registering with a management object, the service aggregator.

Figure 1 shows the object classes involved in the service management activity.

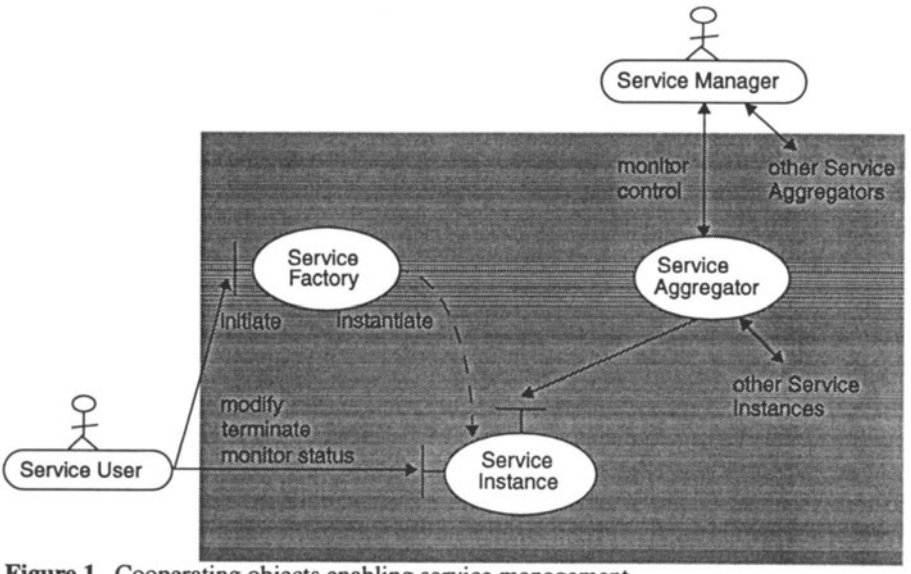

Figure 1 Cooperating objects enabling service management

For each type of service there is a service factory which handles the requests for new service sessions and coordinates the service creation/instantiation process. For example, a service factory for unicast VCs receives a request for a new VC and contacts the necessary controllers (connection managers and routers) in order to set up the new VC. As a result the user who requested the service will receive a handle to use the new VC. The service factory object abstracts the specific scheme used for the service instantiation process.

The service instance represents the status and capabilities of an existing service session. Figure 1 shows the service instance with two types of interface. One is accessed by the service user and represents the control capabilities (status, modify, terminate) a user has once the service session is created; the other is accessed by the management system and represents the management capabilities. The service aggregator has two purposes. First, it provides a single point of access for the service manager to monitor and control the existing service sessions of a particular service type. Second, it provides aggregated or abstracted views of the service instances and allows the manager to manipulate sets of service instances. The service factory and the service aggregator are *persistent* objects. They are created and initiated at service deployment time and generally stay alive as long as the service is provided. The service instance, on the other hand, is a *dynamic* object. It is created when a service session is instantiated by the service delivery system, and it is deleted when the session is terminated.

The model we are proposing is generic in the sense that it describes object classes, their relationships and their interactions for the purpose of making service sessions manageable. In fact the set of operations at the interfaces, both offered to the user and to the manager, are generic for any kind of service.The specific functionality of, say, a service

factory, depends on the particular service. Similarly, the lifetime of service instances differ from service to service in a typical range from minutes to days. Service instances of different types can be related via service composition. For example, a teleconference may be built using a set of unicast and multicast VCs.

An important aspect of our model is that the designer of a service has several degrees of freedom when developing service management functionality. The choices include the selection of the state information and functionality of the service instances, the amount of information kept in the service aggregators and the consistency requirements for the management data in the aggregators. All these factors influence the *cost* of managing the service in terms of design complexity during the development phase and in terms of processing and computational resources needed during the operational phase.

Related Work

The need for making services manageable has been widely recognized. Work on this topic is being pursued within the areas of distributed computing, internet networking and--of course-- telecom systems. Within the field of distributed computing, Schade et al. [SCH96] propose an extension to CORBA IDL, which allows for specifying the management interface of service components in a distributed environment. The authors provide a library for implementing such a management interface. An approach to make services manageable for an ISP (Internet Service Provider) is described in [BHO97]. The authors address the problem of fault diagnosis in a multi-domain environment. Using the concept of service contracts, they suggest an architecture that provides an ISP with management (monitoring) interfaces to service components belonging to the domain of another ISP. In the telecom field, the activity most related to our work is the TINA initiative [TINA95]. TINA also applies an object-oriented methodology for developing both the service delivery and the service management systems. So far, the TINA effort has concentrated on the service delivery system; aspects of service management as presented in this paper have not been addressed until now. The EURESCOM P610 project has recently been launched to develop an architecture for multimedia service management [EUR97]. Because it addresses issues similar to those discussed in this paper, results from this project will potentially influence our research.

3 *mcast*--A MULTIMEDIA SERVICE

mcast is an ATM multicast service based on an object oriented design. It provides the capability of transporting multimedia information from a source to one--or more than one--destination. An *mcast* session belongs to one specific service class, i.e., video, data or voice. (A service class defines the performance characteristics and the performance requirements of a cell flow.) A destination can request to join or leave an *mcast* session. The termination of a session is requested explicitly by the source.

The *mcast* design includes three classes of controller objects: the *mcast* directory service (MDS), the *mcast* router (MR) and the *mcast* connection manager (MCM). The first two classes are global controllers, i.e., they maintain global knowledge about the service. In contrast, the MCM keeps local knowledge (from a host or switch).

The main purpose of MDS is to allow prospective destinations to get information about the currently existing *mcast* service sessions. MDS also keeps partial information about the current graph that represents each *mcast* session for routing purposes. MR computes the route to a prospective destination upon request. It first selects an intermediary node in the tree to act as source in the route.

Figure 2 shows the cardinality relationship among *mcast* controllers. The NodeServer encapsulates the switch capabilities to establish and terminate local unicast connections. It realizes the interface between the service delivery system and the network resources.

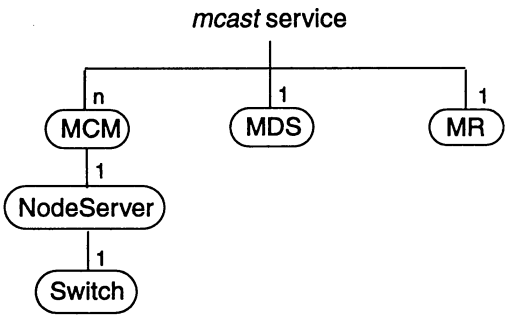

Figure 2 Cardinality relationship among *mcast* controllers

There is one MCM instance for each node (host or switch). The operations offered by MCM are associated with two interfaces: one offered to the service user and the other offered to a peer MCM. The operations at the service user interface are: `initiate`, `join`, `renegotiate`, `leave` and `terminate`. They all refer to one specific *mcast* session. An MCM receiving a `join` request coordinates the reservation of resources by interacting with the peer MCMs along the route to the new destination. Upon a `leave` or a `terminate` request, the interaction among MCMs is hop by hop. This interaction can happen in either direction--from destination to source or vice versa--depending on which controller initiates the process: the source, a destination or a third-party.

To further clarify our design, Figure 3 shows the *mcast* object classes and the way they invoke each other.

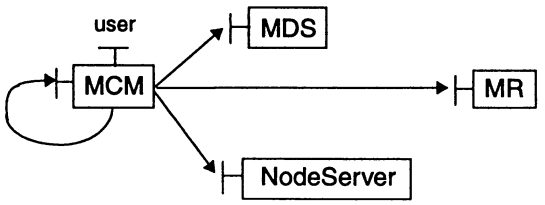

Figure 3 Invocation relationship between *mcast* object classes.

The *mcast* service capabilities are similar to those defined by the ATM Forum [ATMF95] or the IETF [ZHA93]. For instance, *mcast* allows for root, leaf and third-party initiated join and leave requests; contrary to the IETF scheme, an *mcast* session stays alive until it is explicitly terminated. We are considering the following service extensions for *mcast*: (1) allowing more than one source per *mcast* session; (2) providing different QoS to different destinations.

4 MAKING *mcast* MANAGEABLE

Using the model introduced in Section 2, we present a design for managing service instances and aggregations of *mcast* sessions. The management system monitors the set of existing *mcast* sessions and is able to control the system by creating new *mcast* instances, modifying the state of existing instances and terminating instances.

In order to make a service manageable, (1) the service delivery system has to maintain the service instances and (2) the service management system has to be able to access the instances.

To accomplish (1), we need to identify the controller(s) offering the service interface to the user. In our service management model, this interface is provided by the service factory object (creation) and the service instance object (all other operations). In *mcast,* this interface is provided by the *mcast* connection manager (MCM). As described in Section 2, MCM provides the following operations to the user: `initiate`, `join`, `renegotiate`, `leave` and `terminate`. Figure 4 shows an example in which the MCMs, acting as service factories, generate the service instance objects.

To accomplish (2), upon creation, these objects make themselves accessible to the management system by registering with the *mcast* service aggregator.

To refine the design, the following questions need to be answered: What is the state about an *mcast* session that is to be kept in a service instance object? What is the mechanism by which service instances are updated? What is the level of aggregation within the service aggregator? What is the interface between the service aggregator and the service manager?

For our implementation we have defined the following information as the state of a service instance object:

- resource requirements: the service class, i.e., video, data, audio;
- connectivity graph: the size --number of endpoints-- and network resources involved;
- traffic statistics: this characterizes the dynamics of *mcast* sessions in terms of destinations joining and leaving.

Next, the interfaces *a*, *b* and *c* shown in Figure 4 need to be defined. Each of them will affect the interfaces of two controllers.

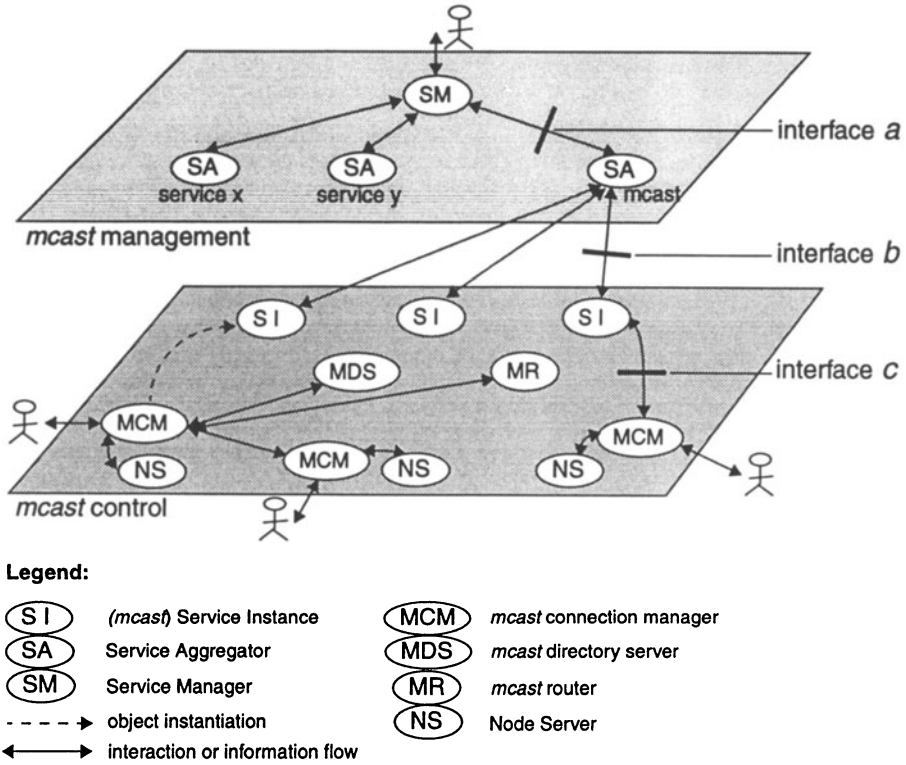

Figure 4 Management of *mcast* service sessions.

4.1 Monitoring service instances

There are two possible ways of monitoring: either the management system polls the system being managed for its state, or the system being managed sends notifications about its state to the management system.

In our implementation of interface *a*, both polling (from service manager to service aggregator) and sending notifications (from the service aggregator to the service manager) are supported, for a single service instance and for a set of instances. The operations defined are:

- for a session: `newSession`, `terminatedSession` as notifications and `getState` as a polling operation (this allows the service manager to obtain the current state of a session);
- for a set of sessions: `numberSessions` and `getState`, as polling operations. In the current implementation a set is specified by enumeration.

Monitoring in *b* and *c* is accomplished via notification messages. In *b* these messages, which travel from the service instances to the service aggregator, are: `newSession`, `modifiedSession` and `terminatedSession`. In *c* the messages are: `modifiedSession` and `terminatedSession`.

4.2 Controlling: affecting the state of service instances

In our implementation the management system is able to perform the following types of operations: `create` a service instance, `modify` one service instance or a set of service instances, and `terminate` one service instance or a set of service instances.

For example, the management system connects a video broadcast to the network. This procedure includes the service manager requesting a new session via the *mcast* service aggregator, which, in turn, contacts an MCM acting as the service factory. As a result, the availability of the video broadcast is advertised through the *mcast* directory service.

The `modify` operation applies to the state of a service instance or a set of instances. Given the above definition for the state of a service instance (resource requirements, connectivity graph and traffic statistics), `modify` can apply to either the requirements or the graph. If the modification request applies to the service class, the request is translated by the delivery system into a `renegotiate` operation. If it applies to the graph, then it is translated into a `join` or a `leave` operation.

5 IMPLEMENTATION

We have implemented the *mcast* service and the management capabilities described in Sections 3 and 4 on two platforms, namely, a) on a high-performance emulation platform, which allows us to develop a software prototype and to study its dynamic behavior and scaling properties in various scenarios [CHA96b], and b) on a broadband testbed running xbind [LAZ96], a CORBA-based multimedia networking platform, on which services are fully implemented and the transport between multimedia devices is realized.

Our requirement is that the implementation design of the service control and management systems can be done in such a way that their code runs on both platforms. Also, the GUI enabling service management functionality by an operator must run on both platforms.

5.1 The emulation platform

Figure 5 shows the building blocks of the emulator platform. The emulated system and emulation support modules consist of a set of objects that communicate by exchanging messages, using functions provided by the simulation kernel. The emulated system module represents the prototype system under evaluation. In our case, this module contains the controllers of the *mcast* service delivery and management systems. Generally, each controller is implemented as a C++ object.

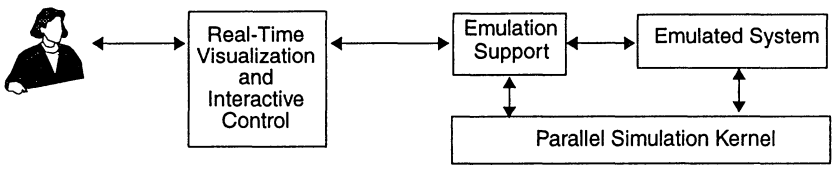

Figure 5 Building blocks of the interactive emulation platform.

The simulation kernel controls the execution of these objects and ensures that messages are processed in the correct order. It is realized as a parallel discrete event simulation (PDES) system, using a window-based synchronization protocol. In order to support real-time visualization and interactive control of the emulated system, the kernel controls the progression of the simulation time, constraining it by the progression of the processor time. The module for real-time visualization and interactive control contains a graphical interface which provides 3-D visual abstractions of the system state.

Both the emulated system and the simulation kernel (coded in C++) run on an SP2 parallel processor located at the Cornell Theory Center (CTC) in Ithaca, New York. The real-time visualization and interactive control module resides on an SGI Indigo2 workstation at Columbia University. It is written using Open Inventor, a 3D graphics tool kit based on Open GL. The emulation support module is distributed on the two machines. These machines communicate through NYNET, an ATM network that connects several research laboratories in New York State.

5.2 The broadband testbed

We use CORBA technology [OMG96] for building the *mcast* service delivery system as well as the management system on the broadband testbed. CORBA offers a flexible object oriented environment for the design of distributed software. The CORBA implementation we use is based on OrbixV2.0 [IONA96].

The broadband testbed is a local ATM network that connects a set of workstations and multimedia devices, such as cameras and microphones. The testbed runs xbind [LAZ96], a CORBA-based open platform for service creation and implementation. xbind defines and implements a collection of CORBA objects called the Binding Interface Base (BIB). The BIB provides access to abstractions of networking resources and multimedia devices. For example, the VirtualDevice interface abstracts the operations that apply to a multimedia stream device; the VirtualSwitch controls the manner in which VPI/VCI pairs are allocated and deallocated for ATM switches and ATM adapter cards. Support for QoS is realized via a set of abstractions that characterize the multiplexing capacity of networking and multimedia resources. The control of the ATM switches is accomplished via the General Switch Management Protocol (GSMP) [NEW96].

Figure 6 gives an overview of the current testbed architecture. At the lowest layer, the ATM LAN connects a set of hosts including multimedia devices (cameras, speakers, microphones and displays). The BIB is at the lowest software layer. *mcast* controllers, located in the layer above, make calls to the BIB interfaces to access the networking resources and multimedia devices. The highest layer contains the management entities.

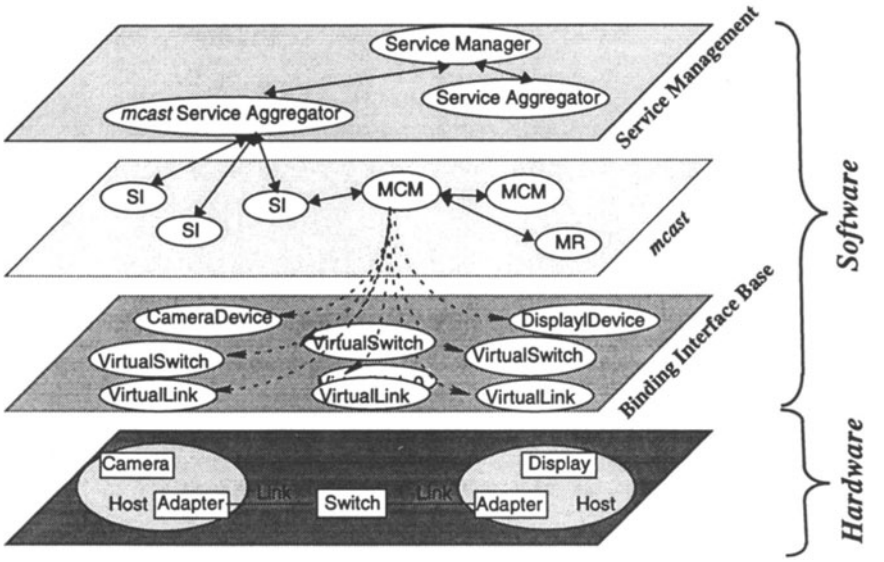

Figure 6 Hardware and software layers in the broadband testbed

5.3 Comments on our implementation

Figure 7 shows the 3D operator interface used in both platforms for the management of the *mcast* service. The network topology is shown at the lowest layer. In it the link states are represented as vertical bars standing on them. Different colors on these bars represent different service classes, such as video, audio and data. The balls above represent *mcast* sessions for which the nodes below act as sources. For the six sessions starting at the node nearest the viewer (bottom left), the multicast trees are shown.

We have developed a scheme allowing us to re-use the implementation design and the code for both platforms. *mcast* controllers and management components are realized as C++ objects, which run on both platforms with minor modifications.

The state of our work is as follows. We have completed the implementation of the service control and management systems on both platforms, and we have validated their functionality for a small network. On the emulation platform, we are currently studying scaling properties of the management system (see discussion); on the CORBA platform, our current work deals with connecting the CORBA implementation to the xbind device and resource abstractions, enabling *mcast* to deliver multimedia transport services end-to-end among multimedia devices.

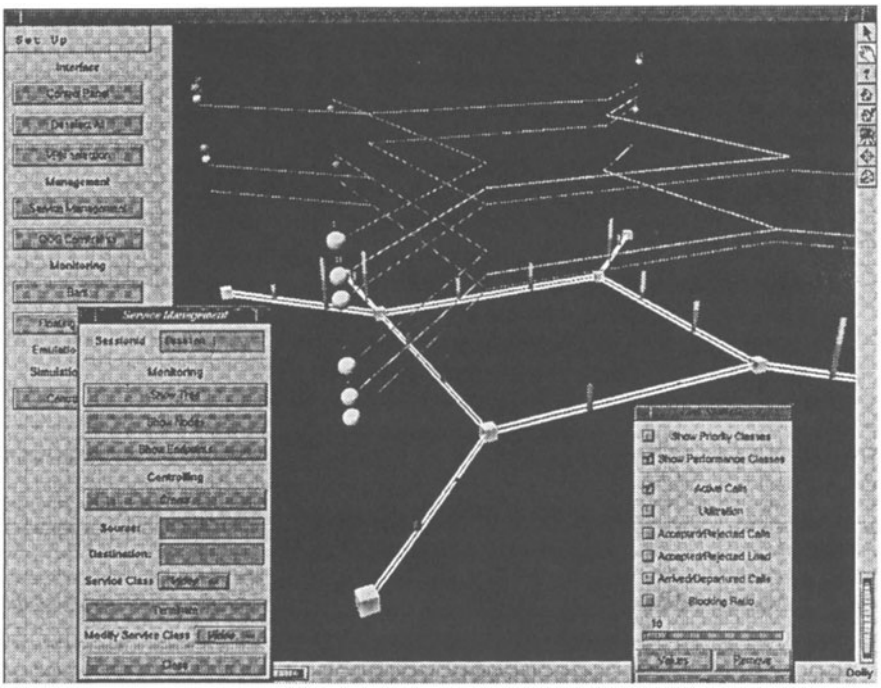

Figure 7 The operator interface displaying the current link load and the current sessions of the *mcast* service. The window in the lower left corner allows the operator to perform service management operations on selected *mcast* sessions.

7 DISCUSSION AND FUTURE WORK

Our implementation proves that the model introduced in Section 2 is applicable to the *mcast* service and indeed makes service sessions manageable. Applying our model to similar services like a point-point VC service or another flavor of a multicast service seems straightforward. Using our model, we are currently designing and implementing a "Virtual Workshop" teleconferencing system and its management on top of the *mcast* service.

Our current research is not so much directed towards applying our model to additional services in order to prove its generic applicability as to use the model as a guide to investigate design approaches to *efficient and scalable management systems*. Efficiency relates to using minimal resources to achieve a set of functional requirements. By scalable, we mean the ability of the system to a) support high rates of service requests, caused e.g., by a high load of the service control system combined with a short average life-span of service sessions, and b) to offer interfaces between service aggregators and service managers that are suited for a large number of concurrent sessions.

As outlined in Section 2, a designer has several degrees of freedom in order to introduce management functionality to a particular service. Consider, for example, the service instance of *mcast*. The state of such an object can simply consist of an identifier of a service session, but, additionally, it can include the end points of the multicast session, or the topology of the multicast tree, or the VCL identifiers of the edges of that tree, etc. Likewise, the functionality of the service instance must include a command to terminate a session, but it can also include one or more of the following commands: query session status, modify session properties, and terminate a specific sink.

Consequently, there is a wide range of possible designs for building management systems for a particular service, from light-weight, low-cost systems with minimal functionality to heavy, high-cost systems with rich functionality. More precisely, we believe that a trade-off analysis must be made, balancing various factors in order to achieve the best combination between low cost, good scalability and rich functionality of the management system for a specific service and a particular environment. There are strong reasons to engineer configurable systems which can be customized at the time of service deployment. Moreover, it may be necessary to allow for dynamic reconfiguration of a system during run-time, in response to changing needs and the availability of resources allocated to management tasks.

In the current *mcast* prototype system, the service aggregator creates a global view by simply making all states of the service sessions available at its interface, and it allows the service manager to execute control operations on a single session or a set of sessions. Based on our experience with the emulation platform, such a design is scalable up to a few dozen service sessions. For large numbers of sessions (10^3 or more), other approaches to the design of this interface are needed. Specifically, such an interface must include operations involving aggregated information on sessions in the sense of condensed or abstracted data. This type of functionality is beyond that of the scoping and filtering mechanisms supported by OSI-based management systems or the interface languages of today's database systems (which are typically based on set and relational calculus).

Acknowledgments

This research was supported by the Department of the Air Force, Rome Laboratory, under contract F30602-95-R-0143.

8 REFERENCES

[ATMF95] ATM Forum. *UNI V4: User to Network Interface specification.* ATM Forum, 1995.

[BHO97] Bhoj, P., Caswell, D., Chutani, S., Gopal, G. and Kosarchyn, M. *Management of new federated services.* IFIP/IEEE International Symposium on Integrated Network Management (IM'97). San Diego, CA, May 1997.

[CHA96a] Chan, M.C., Pacifici, G. and Stadler, R. *Realizing global control in multimedia networks.* IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '96), L'Aquila. Italy, October 1996.

[CHA96b] Chan, M.C., Pacifici, G. and Stadler. R. *Prototyping network architectures on a supercomputer.* Fifth International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, NY, August 1996.

[EUR97] EURESCOM P610 participants. *Providing framework, architecture and methodology for multimedia services management.* IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '97), Australia, October 1997.

[GAM95] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley, 1995.

[HYM91] Hyman, J.M., Lazar, A.A. and Pacifici, G. *Real-Time scheduling with quality of service constraints.* IEEE Journal on Selected Areas in Communications, September 1991.

[IONA96] Iona Technologies Ltd. *The Orbix architecture.* http://www.iona.ie/Orbix/arch/index.html, November 1996.

[ITU92] ITU-T Recommendation M.3010 (1992). *Principles for a Telecommunications Management Network.* ITU, 1992.

[LAZ96] Lazar, A.A., Lim, K.S and Marconcini, F. *Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture.* IEEE Journal of Selected Areas in Communications, September 1996.

[NEW96] Newman, P., Hinden, R., Hoffman, E., Liaw, F.C., Lyon, T. and Minshall, G. *General Switch Management Protocol Specification. Version 1.0.* Ipsilon Networks, Inc., February 1996.

[OMG96] Object Management Group (OMG). *CORBA 2.0/IIOP Specification.* Technical Document ptc/96-08-04, http://www.omg.org/corba/corbiiop.htm, August 1996.

[SCH96] Schade, A., Trommler, P. and Kaiserswerth. *Object Instrumentation for Distributed Applications Management.* Distributed Platforms, Chapman & Hall, London, 1996.

[TINA95] Nilsson, G., Dupuy, F. and Chapman, M. *An overview of the Telecommunications Information Networking Architecture.* TINA 95, Melbourne, Australia, February 1995.

[ZHA93] Zhang, L., Deering, S., Estrin, D., Shenker, S. and Zappala, D. *RSVP: A new resource reservation protocol.* IEEE Network, September 1993.